An Efficient Framework of Using Various Decomposition Methods to Synthesize LUT Networks and Its Evaluation

Shigeru Yamashita Hiroshi Sawada Akira Nagoya

NTT Communication Science Laboratories
2-4, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0237 Japan
Tel: +81-774-93-{5275, 5273, 5270}
Fax: +81-774-93-5285
e-mail: {ger, sawada, nagoya}@cslab.kecl.ntt.co.jp

Abstract— We present an efficient framework for synthesizing look-up table (LUT) networks.

Some of the existing LUT network synthesis methods are based on *functional (boolean) decompositions*. Our method also uses functional decompositions, but we try to use various decomposition methods, which include *algebraic decompositions*. Therefore, this method can be thought of as a general framework for synthesizing LUT networks by integrating various decomposition methods. We use a *cost database file* which is a unique characteristic in our method.

We also present comparisons between our method and some well-known LUT network synthesis methods, and evaluate the final results after placement and routing. Although our method is rather heuristic in nature, the experimental results are encouraging.

I. Introduction

When implementing a combinational logic function using a given technology, the desired function must be decomposed or factorized to smaller functions so that the decomposed functions can fit onto the implementation primitives of the technology. Accordingly, many decomposition methods have been proposed. Most of these methods are based on transforming the algebraic expressions of switching formulas, which we call them algebraic decomposition methods. For example, kernel extraction [1] is an example of one superior method. Such decomposition methods appear to be reasonable in conjunction with the technology mapping phase for standard technology libraries.

To realize combinational logic functions using a lookup table (LUT) based field programmable gate array (FPGA), we must generate an LUT network where each LUT is a special node that can realize any function with K (typically 4 or 5) inputs. Most LUT network synthesis methods can be divided into the following two categories.

The methods in the first category are extended methods for the standard technology libraries:

- First, a logic optimizer performs decomposition and technology-independent optimization. In this phase, algebraic decomposition methods are usually used, and the number of literals is used for the cost considering the standard technology libraries.
- Next, a technology mapper covers nodes to K-input nodes.

In this category, there are state-of-the-art methods such as Chortle-d [2], MIS-pga-delay [3][1] and FlowMap [4]. For the covering phase, optimal algorithms have been developed under specified conditions [2, 4]. However, the intermediate networks before the covering phase often affect the final results; in such cases, the final results are sometimes not so good.

The methods in the second category consist of only one phase: they directly transform primary output functions (not expressions) in terms of primary inputs represented by an ordered binary decision diagram (OBDD) [5, 6]. (Below, we call transformations of functions functional decomposition methods.) Therefore, the final results are not affected by intermediate results, and are usually better than the results of the methods in the first category.

The decomposition form of the functional decomposition methods used in the methods in the second category is limited to a specified form based on Disjoint Decomposition [7]. In some cases, another type of functional decomposition called Non-Disjoint Bi-Decomposition may be more appropriate [8]. However, there is no method that positively tries to utilize various functional decomposition methods including Non-Disjoint Bi-Decomposition in synthesizing LUT networks.

Considering the above discussion, we propose a method with the following properties:

- Various decomposition methods, not only *algebraic* but also *functional* decomposition methods, can be integrated.
- It consists of only a decomposition phase. That is, we do not need to consider the covering effect after the decomposition phase.
- To select the "best probable" decomposition at an intermediate decomposition, a "cost database file" is introduced.

Various decomposition methods, such as Disjoint Decomposition [7], Non-Disjoint Bi-Decomposition [8], Weak Division by Kernels, and Davio Expansion can be integrated into our method. Our method can be thought of as an extension of the methods in the second category and a general framework for

[1] MIS-pga partially uses a functional decomposition method.

0-7803-5973-9/00/$10.00 ©2000 IEEE.

synthesizing LUT networks by integrating various decomposition methods. Although it is rather heuristic in nature, the experimental results are very encouraging.

This paper is organized as follows. In Section II, we briefly explain decomposition methods which are used in our method. In Section III, we propose a method of using various decomposition methods to synthesize LUT networks. We present experimental results in Section V. We mention the features of our method in Section VI. Section VII concludes this paper.

II. Preliminary

We treat a network as a directed acyclic graph (DAG) where each node has a specified internal function with respect to its fanins. If the number of fanins of a node is not more than K, we call the node **K-feasible**.

Our problem is to generate the lowest cost network where all nodes are K-feasible.

The cost of a decomposed network is defined as follows: (the number of nodes in the network) + (W × the levels of the network), where W is the user defined weight.

We consider various decomposition methods to be incorporated into our method. Here we summarize some of them as follows.

Disjoint Decomposition

The disjoint decomposition is the form: $f = \alpha(g_1(X^B), \ldots, g_t(X^B), X^F) = \alpha(\vec{g}(X^B), X^F)$, where X^B and X^F are disjoint variable sets [7]. This decomposition can be found by using the OBDD representing the function of a node to be decomposed [6, 9]. In the previous LUT network synthesis method, $|X^B|$ is limited to K so that each $g(X^B)$ can be mapped into a single LUT. In our method, we prepare this kind of method with $|X^B|$ as 3 up to K because we also consider the covering effect at the same time when decomposing a node, which will be discussed later.

Non-Disjoint Bi-Decomposition

The decomposition form: $f = \alpha(g_1(X^1), g_2(X^2))$, where X^1 and X^2 are not limited to disjoint variable sets, can be effectively found by the method proposed in [8]. For some functions, this decomposition form is better than Disjoint Decomposition [8]. The method can treat an incompletely specified function for f, and represents g_1 and g_2 as incompletely specified functions, which is an advantage of this method. However, if we want to use this decomposition together with Disjoint Decomposition, we need to consider covering nodes at the same time as will be mentioned in Section III because this decomposition produces a two-input node as a root node for a decomposition, which is very different from the case of Disjoint Decomposition. This is one of our motivations to propose the framework in this paper.

Weak Division by Kernels

This is a decomposition method using sum-of-products expressions [1]. The computation time is usually smaller than functional decomposition methods since it is based on the algebraic division of expressions.

Davio Expansion

The Davio Expansion has the following three forms:

- $f = f_0 \oplus x_i \cdot f_2$,
- $f = f_1 \oplus \overline{x_i} \cdot f_2$ and
- $f = \overline{x_i} \cdot f_0 \oplus x_i \cdot f_1$,

where $f_0 = f_{\overline{x_i}}$, $f_1 = f_{x_i}$ and $f_2 = f_0 \oplus f_1$. This decomposition is important because any function can be decomposed by using these expansions.

III. Our LUT Network Synthesis Method

A. Concept of Our Method

Our strategy is based on the following concept. Suppose we have various decomposition methods. We can find the best decomposed network from the search space by considering all of the possible combinations of the decomposition methods and the covering effect. However, performing an exhaustive search for all of the possible combinations is not practical. Therefore, we instead select a "best probable" decomposition at an intermediate decomposition.

If we must think of the covering effect after the decomposition phase, it becomes difficult to determine a "best probable" decomposition at each intermediate decomposition, because the decomposition forms are likely to be different between some of the decomposition methods. Thus, it is difficult to predict the covering effect when the decomposition is being done.

With this in mind, we evaluate the "cost" of a decomposition form with the following strategy.

- We evaluate the cost of a decomposition including the covering effect at the same time.
- We predict the cost of nodes whose supports are more than K by using a "cost database file," which describes decomposition costs of functions from previously designed results.

As a result, we can utilize various decomposition methods in our method.

B. Outline of Our Method

The overall procedure of generating a network whose nodes are all K-feasible is as follows.

Step 1: Construct an initial network that has only primary output nodes whose internal logics correspond to the primary output functions (in terms of primary inputs) of the given specification.

Step 2: As long as there remains a node that is not K-feasible, we decompose the node by using a selected decomposition method. How to select a "best probable" decomposition is mentioned in Section III-C.

For a node to be decomposed, we prepare both sum-of-products expressions and functions by OBDDs for the internal logic in order to utilize both algebraic and functional decomposition methods.

C. How to Select a "Best Probable" Decomposition

We characterize a decomposition form of the various decomposition methods used in our method as follows: a decomposition form of a node n is characterized as a node n', which is a replacement of n, and newly introduced nodes $n_1, \cdots n_p$, which are fanins of n'. We can treat most decomposition methods in this form. Fig.1(b) shows an example of this for Bi-Decomposition based methods. We call the set of nodes introduced at the decomposition "DecompArea" (the dotted rectangle in Fig.1(b)).

In our decomposition form, we do not share common functions between some functions. This is because we bravely omit sharing functions in order to uniformly treat various decomposition methods. However, this can be considered as an extension of our method and will be mentioned in Section IV-A.

We select a "best probable" decomposition form of a node at Step 2 in our method by evaluating the "cost" of the decomposition. Since we want to treat various decomposition methods, we consider the case where the number of fanins of a node in the DecompArea is less than K. For example, the number of fanins of n' is two when a decomposition method based on Bi-Decomposition is used. Such a node may be merged into a node not in the DecompArea. Since our strategy does not perform the covering phase after the decomposition phase, we try to merge such a node, which is at the boundary of the DecompArea, into a node not in the DecompArea to form a newly merged node if the merged node is still K-feasible as shown in Fig.1(c). In this example, n' and n_2 can be merged into other nodes, so we do not consider them in the decomposition cost. Accordingly, the cost evaluation after the merging of the nodes simultaneously includes the covering effect.

For the DecompArea after the merging (the dotted rectangle in Fig.1(c)), our cost is defined as:
cost of a decomposition =

$$\{\sum_{n_i \in DecompArea} CostLUT(n_i)\} + W \times \{\max_{n_i \in DecompArea} LEV(n_i)\},$$

where W is the user defined weight. $LEV(n_i)$ is recursively defined as follows, and it becomes 0 for a primary input node.

$$LEV(n_i) = \{\max_{n_j \text{ is a fanin of } n_i} LEV(n_j)\} + CostLEV(n_i).$$

$CostLUT(n_i)$ and $CostLEV(n_i)$ denote the predicted numbers of K-LUTs and the levels for implementing the internal function of n_i, respectively. They become 1 for a

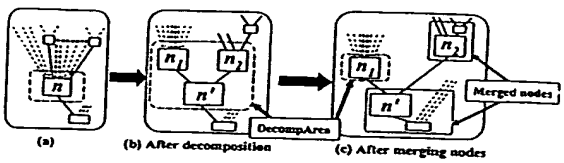

Fig. 1. Decomposition form of a node.

K-feasible node. However, we cannot know the precise values of $CostLUT(n_i)$ and $CostLEV(n_i)$ if n_i is not K-feasible. Therefore, we determine their values by looking up a cost database file as mentioned in Section III-D.

D. Decomposition Cost

In our cost strategy, at first we prepare a cost database file which stores the statistical relationships between some parameters characterizing the output function (in terms of the primary inputs) of a node n_i, and $CostLUT(n_i)$ and $CostLEV(n_i)$. In the present implementation, we use the number of supports of the function, and the number of cubes and literals in an expression for the function.

We generated a cost database file as shown here, but clearly this is not the only method.

- We make a first cost database file in which $CostLUT(n_i)$ and $CostLEV(n_i)$ take the same value as follows.

  $$\begin{cases} 1, \text{if } n_i \text{ is } K\text{-feasible} \\ (\textit{the number of fanins of } n_i) - K + 1, \text{ otherwise} \end{cases}$$

  This value is taken from [5]. We do not consider the number of cubes and literals in this first cost database file.

- Using the first cost database file, we generate various networks by our method. We then make a second cost database file in which each entry describes a statistical relationship between the above three parameters for the output function of each node in the decomposed networks, and the number of transitive fanins of the node and levels of the node, which correspond to $CostLUT$ and $CostLEV$, respectively.

TABLE I
A COST DATABASE FILE

supports	cubes	literals	$CostLUT$	$CostLev$
...	...	...	...	...
7	24	113	7	3
...	...	...	...	...

For example, if we need 7 LUTs and 3 levels to implement a function whose supports, and cubes and literals are 7, 24 and 113, respectively, by using the first cost database file (it actually happened in our experiments), we get an entry in the second cost database file as shown in Table I. We think the second cost database file is more accurate than the first cost

database file because the latter predicts that both $CostLUT$ and $CostLEV$ for the function are 3 ($K = 5$), which is quite different form the actual results. The second cost database file can be thought of as a feedback from the previously designed results. Actually, we usually obtain better results with the second cost database file than we do by using the first cost database file.

With the cost database file, we calculate $CostLUT(n_i)$ and $CostLEV(n_i)$ as follows.

- Calculate three parameters from the internal function of n_i.
- Find the values of $CostLUT(n_i)$ and $CostLEV(n_i)$ in the entry that produces the best fit for the three parameters in the cost database file.

In most of the previous logic synthesis methods, the cost of a function is usually measured only by the number of supports of the function or literals in the logic expression of the function. We can use the both parameters in our method.

We can generate the third cost database file from the second cost database file in the same way. With the third cost database file, we sometimes obtain better results than we do by using the second cost database file.

IV. An Extension of Our Method

A. Sharing Sub Functions

As previously mentioned, our strategy takes little account of the sharing of common functions, which sometimes dramatically reduces the network cost. Therefore, we plan to add the following operation to the decomposition methods that are also used as decomposition methods at Step 2: When a node n_i is decomposed, we check whether an existing node n_j can be used for the node. This can be accomplished by dividing the expression of n_i by the expression of n_j, which is called algebraic resubstitution. This can also be done by utilizing the boolean resubstitution and the support minimization technique proposed in [6]. Note that we can adopt the above operation as a decomposition method in our framework if we do not consider n_j in the decomposition cost.

In our framework, we can also prepare another operation to share common functions: after all decompositions, the minimization method proposed in [11] is performed to replace the output of a node with that of another node.

B. Speeding Up the Framework

We believe that some decomposition methods had better be applied first if possible. For example, a simple disjunctive decomposition usually provides good decomposition forms that can be found relatively fast [10]. Such decomposition methods should be applied before the decomposition of a node at Step 2. We expect that this process will sometimes reduce the total computation time.

Another technique of speeding up the framework is to independently checking each decomposition method at Step 2. Indeed, we can perform decomposition methods in parallel on different processors, and this reduces the computation time. The idea is as follows. If a decomposition method takes a much longer time to decompose some functions (or expressions) than other decomposition methods, the decomposition result is usually worse. Therefore, we abandon some of decomposition methods that take too much time in a parallel implementation without sacrificing the quality of our results. This dramatically reduces the computation time. As processors are getting cheaper and cheaper, an implementation in parallel becomes more attractive for our method.

V. Experimental Results

A. Evaluation of Various Implementations

We can get various results from various implementations of our method; for example, the implementation varies depending on the types of decomposition methods that are integrated and the cost database file that is used. All of our results shown in this section were produced by an implementation using Davio Expansion with each variable, Disjoint Decomposition with $|X^B|$ as 3, 4 and 5, and Non-Disjoint Bi-Decomposition.

In our cost strategy, we originally expect the followings features:

- The results obtained with the $(n+1)$-th cost database file are usually better than those with the n-th cost database file.
- We can control the trade-off between network levels and the number of nodes by using the user defined weight W.

We performed experiments with the first, second and third cost database files and $W = 0.5$, 1 and 2.0. From a comparison of the results, we could not find the above features but instead the following features:

- The results with the second cost database file and $W = 1$ are usually the best. This means that we cannot expect the third cost database file to always be better than the second cost database file.
- If W is larger, the levels usually becomes smaller. However, changing W seemed to have no effect on the number of nodes.

From the above, we do not consider our current cost database files to be robust. However, the differences between the various cost database files and W were not so large, and all results were thought to be good enough, as we will see in the following sub-sections.

B. Comparison Before Placement and Routing

Table II compares the mapping results for 5-LUT networks between our method and several of the well-known level-optimized LUT network synthesis methods. Our results were obtained with the second cost database file and $W = 1$.

The sub-columns "#lut" and "#lvl" show the numbers of 5-LUTs and network levels, respectively. The sub-column

TABLE II
COMPARISON OF MAPPING RESULTS FOR 5-LUT NETWORKS

circuit name	ALTO[12]		mispga-d		chortle-d		FlowMap-r		BoolMap-D[5]		Ours		
	#lut	#lvl	#lut	#lvl	#lut	#lvl	#lut	#lvl	#lut	#lvl	#lut	#lvl	CPU
5xp1	19	2	21	2	26	3	23	3	13	2	11	2	0.33
9sym	7	3	7	3	63	5	61	5	7	3	5	4	0.55
alu2	61	6	122	6	227	9	148	8	43	4	33	4	5.44
alu4	259	8	155	11	500	10	245	10	268	7	85	7	77.33
apex4	-	-	-	-	1112	6	-	-	-	-	302	4	32.67
apex6	229	4	274	5	308	4	232	4	189	4	161	4	691.89
apex7	77	4	95	4	108	4	80	4	78	3	61	4	204.98
clip	33	3	54	4	-	-	-	-	-	-	11	3	2.39
count	47	3	81	4	91	4	73	4	42	2	30	4	732.5
duke2	156	4	164	6	241	4	187	4	193	5	150	4	162.75
f51m	15	3	23	4	-	-	-	-	-	-	10	3	0.34
misex1	14	2	17	2	19	2	15	2	15	2	10	2	0.22
misex3	251	6	-	-	-	-	-	-	-	-	166	6	196.64
rd73	8	2	8	2	-	-	-	-	-	-	6	2	0.21
rd84	13	3	13	3	61	4	43	4	10	2	7	3	0.54
sao2	38	3	45	5	-	-	-	-	-	-	21	3	3.56
vg2	26	3	39	4	55	4	38	4	30	4	21	4	120.16
z4ml	5	2	10	2	25	3	13	3	5	2	5	2	0.13
ALTO	1258	61									793	61	
mispga-d			1128	67							627	55	
chortle-d					2836	62					881	48	
FlowMap-r							1158	55			579	44	
BoolMap-D									893	40	579	44	

"CPU" indicates the CPU run-time (sec.) on a Sun Ultra 2 2200. To compare our results with other results, we show the total numbers for the same circuits in the lower part of the table. The shaded numbers indicate the best results. Our framework appears relatively good in the comparison. We think one reason is that Non-Disjoint Bi-Decomposition sometimes provides good decompositions. Our method sometimes needed a long computation time, which we do not think is a very serious problem, as mentioned in Section IV-B.

C. Comparison After Placement and Routing

We have incorporated the proposed method into PARTHENON [13], which consists of a simulator and synthesizers for a hardware description language SFL (Structured Function description Language).

To evaluate the integrated system, we compared the following two logic synthesis flows.

Using the mapping method in Max+plus II

Step 1 Convert the file format and perform the logic synthesis at the technology independent level (including logic reduction) by PARTHENON, and output the result to MAX+plus II, which is the development system for Altera devices.

Step 2 Perform the technology mapping for the Altera FLEX8000 series [14] by MAX+plus II.

Step 3 Perform placement and routing by MAX+plus II.

Using our mapping method

Step 1 Convert the file format by PARTHENON.

Step 2 Our method is called from PARTHENON system to perform the technology mapping for 4-LUT networks[2]. Then PARTHENON outputs the result with the mapping information[3] to MAX+plus II.

Step 3 Perform placement and routing by MAX+plus II.

The two flows are different depending on the method used to generate LUT networks, our proposed method or MAX+plus II.

Table III shows the results after placement and routing. "#LE" and "Delay" show the numbers of logic elements and the delay values (ns) for the longest paths in the final results, respectively. From the table, we can see that our method also has a good effect on the final results after placement and routing.

VI. FEATURES OF OUR METHOD

The proposed method has the following features.

- Various decomposition methods can easily be integrated into our method. If a new decomposition algorithm has been developed, we can easily check its effectiveness in our framework.
- We can get various results from various implementations of our method. Therefore, we are able to obtain various

[2]A logic element of FLEX8000 has one 4-input, 1-output LUT.

[3]An LCELL primitive in MAX+plus II can be used to attach the mapping information.

TABLE III
COMPARISON OF FINAL RESULTS FOR ALTERA FLEX8000

Base tool	PARTHENON (synthesis & converter)			
Mapping	MAX+plus II		Ours	
Place & Route	MAX+plus II			
circuit name	#LE	Delay	#LE	Delay
5xp1	46	27.2	15	17.5
9sym	35	44.4	8	18.6
alu2	135	40.2	97	32.7
alu4	715	87.4	420	75.2
apex4	1447	70.6	691	46.4
apex6	230	38.6	258	47.6
apex7	113	43.0	140	27.7
clip	33	29.3	20	24.4
count	41	31.2	36	32.9
duke2	322	70.7	261	61.1
f51m	42	26.7	13	20.9
misex1	23	27.1	14	16.3
misex3	604	76.7	313	67.5
rd73	29	32.8	7	18.6
rd84	37	34.5	14	26.7
sao2	80	35.5	37	19.6
vg2	71	33.3	55	36.1
z4ml	10	17.4	6	16.0
Total	3922	773.9	2405	604.9

decomposed networks for a given specification, and can explore a large design space.

There are some interesting features in our cost strategy. It is natural for a "bad" entry (which we think has a bad effect on our cost strategy) to be generated in our cost database file from a "bad" node for which abnormal (unexpected) number(s) of nodes or (and) levels were used in previously designed networks. In our experiment, we ignored a "bad" entry in the cost database file. However, it was interesting that when we resynthesized a "bad" node, the numbers of nodes and levels for the node were reduced at times to normal values in our cost database file. We think this feedback to the resynthesis is one of the advantages of our framework. In other words, the cost of the network was sometimes reduced by resynthesizing the output of "bad" nodes.

VII. CONCLUSION

We have proposed an efficient method for synthesizing LUT networks. In our method, we successfully integrated many decomposition methods that are not only algebraic but also functional. Our method can be thought of as a general framework for synthesizing LUT networks by integrating various decomposition methods.

Currently, our framework cannot treat large networks because some of functional decomposition methods cannot treat large functions. In the future, therefore, we would like to improve the framework by incorporating it with appropriate network partitioning methods, and to extend it using the techniques presented in this paper.

REFERENCES

[1] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: a multiple-level logic optimization system," *IEEE Trans. CAD*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[2] R. J. Francis, J. Rose, and Z. Vranesic, "Technology mapping of lookup table-based FPGAs for performance," in *Proc. ICCAD*, pp. 568–571, Nov. 1991.

[3] R. Murgai, N. Shenoy, and R. K. Brayton, "Performance directed synthesis for table look up programmable gate arrays," in *Proc. ICCAD*, pp. 572–575, Nov. 1991.

[4] J. Cong and Y. Ding, "An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," in *Proc. ICCAD*, pp. 48–53, Nov. 1992.

[5] C. Legl, B. Wurth, and K. Eckl, "A boolean approach to performance-directed technology mapping for LUT-based FPGA designs," in *33rd ACM/IEEE Design Automation Conference*, pp. 730–733, June 1996.

[6] H. Sawada, T. Suyama, and A. Nagoya, "Logic synthesis for look-up table based FPGAs using functional decomposition and support minimization," in *Proc. ICCAD*, pp. 353–358, Nov. 1995.

[7] J. P. Roth and R. M. Karp, "Minimization over boolean graphs," *IBM Journal*, pp. 227–238, Apr. 1962.

[8] S. Yamashita, H. Sawada, and A. Nagoya, "New methods to find optimal non-disjoint bi-decompositions," in *ASP-DAC '98*, pp. 59–68, Feb. 1998.

[9] Y.-T. Lai, M. Pedram, and S. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," in *30th ACM/IEEE Design Automation Conference*, pp. 642–647, June 1993.

[10] H. Sawada, S. Yamashita, and A. Nagoya, "Restructuring logic representations with easily detectable simple disjunctive decompositions," in *Proc. of the Design, Automation and Test in Europe (DATE'98)*, pp. 755–759, Feb. 1998.

[11] S. Yamashita, H. Sawada, and A. Nagoya, "A new method to express functional permissibilities for LUT based FPGAs and its applications," in *Proc. ICCAD*, pp. 254–261, Nov. 1996.

[12] J.-D. Huang, J.-Y. Jou, and W.-Z. Shen, "An iterative area/performance trade-off algorithm for LUT-based FPGA technology mapping," in *Proc. ICCAD*, pp. 13–17, Nov. 1996.

[13] Y. Nakamura, K. Oguri, A. Nagoya, M. Yukishita, and R. Nomura, "High-level synthesis design at NTT systems labs," in *Proc. of the Synthesis and Simulation Meeting and International Interchange*, pp. 344–353, 1992.

[14] Altera Corporation, *Data book*, 1993.